English | Español

TUTORIAL 1.1 – Introducción

Este primer tutorial TIENE QUE SER LEÍDO. Trata específicamente sobre requerimientos de directorio y archivo para usar el motor (HPL). Memoriza esto antes de comenzar a crear modelos, mapas y todo lo demás.

Cuando tratamos con directorios y archivos para el directorio HPL hay un par de factores importantes que debemos saber. ¡HPL no se fija en los directorios! Esto significa que si se colocan dos archivos con el mismo nombre y extensión en lugares distintos, HPL no será capaz de notar la diferencia y simplemente usará el primero que encuentre.

¡Por ello es muy importante siempre tener nombres de archivos únicos!

¿Te parece malo? Es una característica que permite mover archivos y facilita la localización de estos cuando se escriben scripts y archivos de juego específicos. Esto significa que si quieres escuchar un sonido en el fuego solo debes nombrarlo en ves se escribir la ruta exacta al archivo.

HPL no busca en todos los directorios por si mismo, debes decirle a HPL que directorios debe cargar. Esto se hace con el HPL Helper que edita el archivo ”resources.cfg”. Para crear un tutorial para esto, vamos a usar el HPL helper para editar el archivo “resources.cfg”.

Primero vamos al directorio ”redist” y vamos a crear un nuevo directorio llamado ”misCosas”. Inicia el HPL Helper y de inmediato deberías estar en la pestaña “settings”, esta es la parte del HPL Helper que queremos usar. Haz click en ”Add” y encuentra el directorio que acabas de crear para agregarlo a la lista. Si se hace correctamente, deberías ver un “misCosas” al final de la lista.

Vamos a crear más directorios. Primero mueve el directorio “tutorial” dentro del directirio “redist”, queremos hacer esto porque contiene archivos que queremos usar para nuestros tutoriales. Usando el HPL Helper agregamos los siguientes directorios:

tutorials *tutorials/tutorial_1* tutorials/tutorial_2 *tutorials/tutorial_3* tutorials/tutorial_4 *tutorials/tutorial_5*

Tu lista de directorios de recursos en el HPL helper debería ahora listar estos directorios adicionales. Cuando estñe terminado presiona el botón “Save to File” para guardar los cambios. Para saber so lo hiciste bien trata de abrir un modelo de pueba: Ve a la pestaña ”Models” en el HPL Helper y selecciona “Brwse”. Navega hasta “tutorial_1” y selecciona ”tut_woodbox.dae”, prueba con el botón “view”. Si todo está bien, deberías poder ver una linda caja de madera en el visor y si das click en él deberías ser capaz de interactuar con ella.

Con esto concluye el tutorial de directorios.

TUTORIAL 1.2 – Creando tu primer modelo

El uso de HPL para crear juegos requiere tanto conocimiento y tiempo como cualquier otro motor de juegos avanzado dedicados a la creación de contenido/modelos. Lo que hace diferente a HPL del resto es que no utiliza un editor de mapa en específico, sino que en vez de eso utiliza editores 3D comunes.

El único requerimiento es que el editor 3D soporte el formato COLLADA, http://collada.org. A continuación veremos una pequeña introducción a la creación de un modelo básico y como hacerlo funcionar en el juego; después de eso crearemos un nivel que consista de una habitación haremos que la habitación contenga el modelo creado. Este tutorial utiliza texturas ya hechas, las explicaciones sobre las texturas vendrán después.

Para empezar, busca un módulo/extensión/librería para editar que tenga soporte para COLLADA, instala e inicia tu editor, asegurandote de que es capaz de importar y exportar archivos collada. Es importante que actives la opción “Exportar polígonos como triángulos” para la exportación en collada.

Primero, en tu editor 3D establece las unidades a 1 metro. HPL-Engine utiliza el sistema métrico decimal para todas las unidades.

Crea un cubo básico de 1 metro cúbico y colócalo en el centro del proyecto. El cubo necesitará una textura, para facilitarte las cosas tenems una textura lista para usar.

Se encuentra en ”tutorials/tutorial_1/tut_woodbox.jpg”

Agrega la textura al cubo sin renombrar o mover la textura. Ahora deberías tener un lindo cubo con una superficie de madera. Guarda el proyecto en la ubicación que quieras, este archivo de trabajo puede ubicarse en cualquier parte que desees.

Vamos a probar el modelo y a probar si funciona en el juego. Exporta el proyecto como un archivo collada y guarda el archivo en tu directorio ”misCosas” con el nombre de ”misCosas_cajaMadera.dae”. Ahora usa el HPL Helper para abrir el archivo usando la pestaña ”Models”.

Si todo va bien deberías poder ver una caja de madera en este momento. ¡Si tratas de moverla no funciona! Si recuerdas el tutorial anterior, esto si funcionaba con aquella caja, pero no funciona con esta porque el modelo que creamos no tiene un colisionador, esto es algo que debemos agregarle.

Vayamos de regreso al editor 3D y haz una copia de tu caja de madera. No la muevas, déjala exáctamente en la misma posición que el cubo original. Ahora vamos a renombrar el nuevo nodo como ”_collider_box_1”. Lo que hacemos es agregar al juego un ”collider” del tipo ”box” llamado ”1”. El “_” inicial actua como pista para que el motor reconozca que su geometría es especial, los “_” restantes actúan como separadores para las diferentes propiedades.

Distintos tipos de colisionadores están disponibles, box (caja), sphere (esfera), cylinder (cilindro) y más. Para la lista exácta mira el capítulo HPL-Engine© Content Creation. Modelos más complejos pueden tener múltiples colisionadores diferentes, puedes usar tantos como quieras para hacer que la detección de colisiones sea lo más detallada posible, pero la idea también es hacerlo lo más simple posible.

En tu proyecto aora deberías tener un nodo que es la caja original y un nodo que es el nuevo colisionador. Exporta el modelo nuevamente y abrelo usando el HPL Helper. Si lo hiciste correctamente ahora deberías poder interactuar con la caja.

No entraremos en más detalles aquí, pero es bueno siempre agrupar tus figuras con sus colisionadores. Más adelante podrás crear modelos que tengan disintas partes. Digamos que creas una puerta en la que quieres que el marco se quede estático, pero que la puerta sea un objeto que el jugador pueda abrir y cerrar. Para que esto suceda debes hacer que el marco y la puerta sean objetos

distintos en tu editor 3D y agruparlos con sus respetivos colisionadores.

TUTORIAL 1.3 – ENTITY FILES

When you have a model finished you might want to add specific properties to it. Such as what type of material is the object, how much it weighs, how you interact with it and does it collide with the player? To do this you'll need to create an entity file, this is basically a regular text file containing all the information, an XML file to be specific.

To start off lets take a look at an existing entity file. Using a regular text editor (preferably wordpad or better, not notepad and never word or any other office like suite.) open the ”tut_woodbox.ent” file located in tutorials/tutorial_1.

This is a quite simple entity file, let’s take a look at what it contains.

```
<MAIN
    Name="tut_woodbox"
    Type="Object"
    Subtype="Normal"
/>
```

These are quite unimportant. For simplicity always put the name of the file in Name=””. Type and Subtype are not often changed. Type is ”Object” for most models, if you make a model that is going to be picked up, say the dynamite in Penumbra, you change it to Type=”Item”. Subtype is so far not used at all except as Normal. For details on different Types view the 6 Entity files document.

```
<PHYSICS
    SubName = "pCubeShape1"
    Collides = "true"
    HasPhysics = "true"
    Material="Dyn_Wood_Box"
    Mass="30"
    InertiaScale = "1 1 1"
    AngularDamping = "0.1"
    LinearDamping = "0.1"
    BlocksSound = "false"
/>
```

Most of these are self-explanatory. The SubName is important, is should always be the name of the mesh. It's of extra importance when you have a model with different parts, otherwise the engine wont be able to tell, for example, what part of a door that is static and not.

Material=”” says what material the object it, this is used to give it friction, sounds and such material specific things. A list of materials can be found in HPL Helper under the ”Materials” tab, by the drop down menu ”physics material”.

Angular and Linear Damping, lets play with them a little. Using HPL Helper, under the ”Models” tab open the ”tut_woodbox.ent” file. This will open the tut_woodbox.dae file BUT also give it the specific properties from the ent file. Now spin the box around a little, throw it around and notice how it

behaves. Go back to the text editor and edit the entity file to have AngularDamping = “1” and LinearDamping = “1” and save the file. Go back to HPL Helper and open the the file again, now if you bounce and spin the box around you should notice that it wants to spin and move less/slower. This is because we raised the damping factor. As always more exact details on everything can be found in the HPL-Engine© Content Creation.

```
<GRAPHICS
    ModelFile = "tut_woodbox"
    CastShadows = "true"
/>
```

ModelFile =”” this is the name of the model (.dae) file the entity file is setting properties for. This means that you can create one model and then you can create several different entity files, each with different properties but they all use the same model.

```
<GAME
    InteractMode = "Push"
/>
```

Here you set different behaviours for the model. Push means that if you have the wood box in the game, you will only be able to push it around. If you change to ”grab” you will be able to pick it up and carry it around. If it's a door you use ”move” and that will give that function of being able to nicely move objects in a realistic manner with your mouse. These are typical doors, drawers, valves, sticks and so forth. This are can also contain several other settings, as always HPL-Engine© Content Creation and 6 Entity files are the places to search for more details.

The easiest way to get started with entity files it to always make a copy of an existing entity file and use that as a base for your new objects. Select an entity file for an object similar to the one you have created. In a long term perspective it is important to eventually understand what everything is in the entity file though.

So lets create an entity file for our ”mystuff_woodbox.dae” file. Start by copying the ”tut_woodbox.ent” file to your mystuff directory. Name it ”mystuff_woodbox.ent” and open it in a text editor.

```
<MAIN
    Name="tut_woodbox"
    Type="Object"
    Subtype="Normal"
/>
```

Change Name to mystuff_woodbox.

```
<PHYSICS
    SubName = "pCubeShape1"
    Collides = "true"
    HasPhysics = "true"
    Material="Dyn_Wood_Box"
    Mass="30"
```

```
    InertiaScale = "1 1 1"
    AngularDamping = "0.1"
    LinearDamping = "0.1"
    BlocksSound = "false"
/>
```

Make sure SubName is the name of your mesh in your 3D editor. To test having a different material, replace ”Dyn_Wood_Box” with ”Metal”.

```
<GRAPHICS
    ModelFile = "tut_woodbox"
    CastShadows = "true"
/>
```

Change ModelFile to ”mystuff_woodbox”. If you wish set CastShadows to false if you do not want the model to cast a shadow.

```
<GAME
    InteractMode = "Push"
/>
```

Set this to ”Grab” instead of ”Push”.

Now save the file and open it in HPL Helper. If everything works OK, you should see your wood box just as before. The only difference is that it should fall down to the ground by itself, if you open a .dae file directly it stays in the air until you interact with it.

Take not that in HPL Helper the InteractMode for Push and Grab makes no difference, this is only working when you have the object in the game. Which leads us to, of course, how to get the model into the game! For this to happen there are 1 more thing we need to do and that is the next chapter.

TUTORIAL 1.4 – REFERENCES

References are files that you create and import into your levels to tell the engine what entity it should load, where it should be loaded and what it should be named.

To create a reference open your ”mystuff_woodbox” project. This should be your work project and not the .dae file, but it really does not matter which.

Delete everything in the project EXCEPT the original mesh, the actual wood box. Rename the node to ”_ref_mystuff_woodbox_woodbox1”. As you might have guessed this is quite similar to how you created a collider earlier, ”_ref” this is the string telling the engine that it's a reference, ”mystuff_woodbox” is telling the game where to find the file and finally ”woodbox1” is what the object will be name in the game. The name is used when you for example write scripts.

Do not save the project, only export a new collada file. You can name it “ref_mystuff_woodbox.dae”. Now you have a reference file, but to see it in the game we need to create a level. But before moving on to Tutorial 2, where we create a level, we shall have a serious talk about exactly how reference files works.

The actual reference file is not important AT ALL. It's the ”_ref_mystuff_woodbox_woodbox1” that's important to get in the game. Where you store the reference file is of no importance, the reference file is only used by you to place objects correctly in the game. As far as the game is concerned you can use any object you want as long as you name it correctly.

If you want to put a character in the game you do not need to have an actual character model as your reference, it works just as well with a cube as long as it is named ”_ref_characterfile_charactername”. But for you to know exactly where you place it in a level it's logical to use the exact character model as your reference, as well as being able to know what the reference is simply by looking at it. Therefore always create a reference that is an exact copy of the file you are referencing to.

The other important thing to know about references are: They point the game to the ENTITY file and not the MODEL file. When the game loads a level it finds all the _ref named objects and it then loads the entity files that the references points to. When loading the entity file it gets all the specific properties for the model and, as you might recall, the entity file contains the name of the model it sets it's properties for. Leading to the model being loaded into the game with the specific properties.

Now, lets create that level of ours so we can finally test a model in the game.